第3部 本格的な人工社会をめざす

第21章：エージェントを空間上で複雑に動かす

❑21.0 動かし方にはいろいろあります

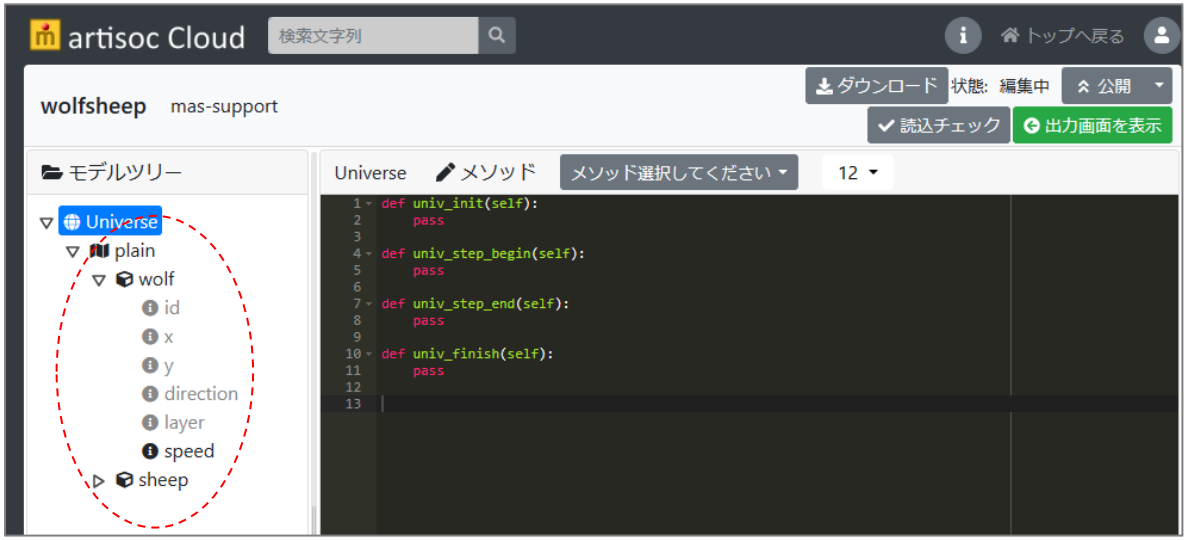
- 羊を追いかける狼をモデル化します。
- 「ループしない」空間にも慣れましょう。
- 狼にも羊にもさまざまな動きをさせてみます。
- 組み込み関数を多用します。
- 状況に応じてこの章のさまざまな技法を利用して下さい。

❐21.1 動かし方のレパートリー

- マルチエージェント・シミュレーションにはいくつかのタイプがあります
  - エージェントが動かないモデル
    - セル・オートマトン：格子状に並ぶ、近傍のエージェント同士が相互作用
    - グラフ（ネットワーク）：１カ所に静止
  - エージェントが動き回るモデル
    - 人工社会：飛ぶ鳥モデルなど
- 空間上でエージェントをさまざまに動かす技法を紹介します
  - 動かし方のレパートリーを覚えよう

❑21.2 モデルの大枠

- 狼と羊のモデル
  - 新規モデル作成　モデル名：wolfsheep
  - Universeの下にplain空間（100 × 100、「ループする」）を生成
  - 空間にエージェントwolfとsheepを追加
  - エージェントwolfには、1ステップあたりの移動速度を表す変数speedを追加
  - マップ出力設定でエージェントwolf、sheepが二次元マップに出力されるように設定



❑21.3 速さや方向を変えながら動く

- エージェントがステップ毎に速度（速さや方向）を変化させながら移動する

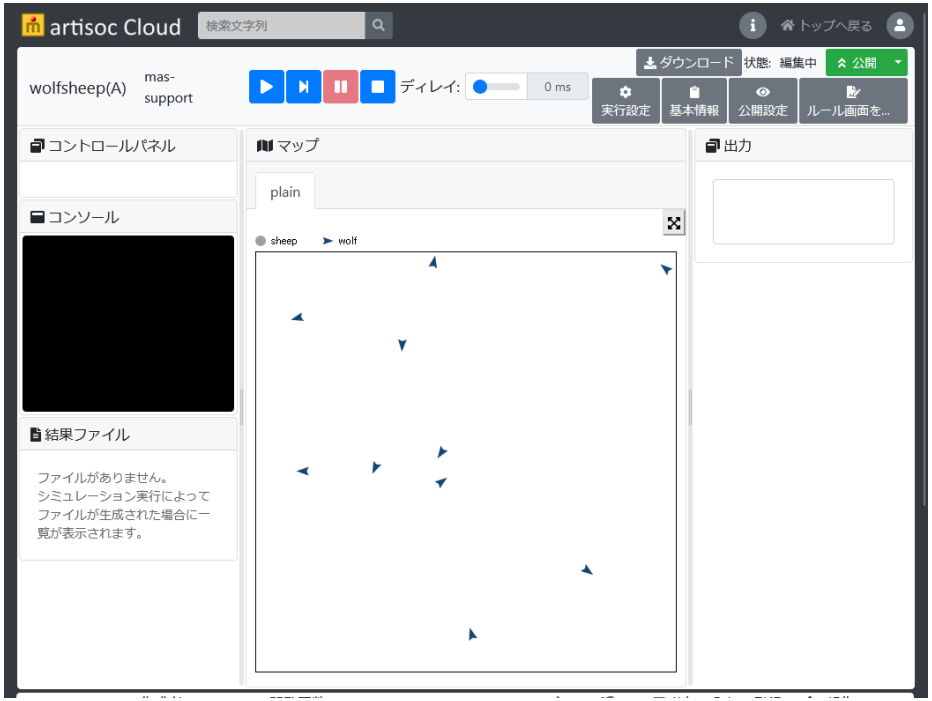
- wolfの動き
  - wolfの数：10
  - speed：0.0から3.0まで徐々に加速（1ステップに0.1ずつ）
  - 初期位置：ランダム
  - ステップ毎に向きを変えながら動く：左右に30度の範囲でランダムに体の向きを変える

```
def univ_init(self):
    create_agt(Universe.plain.wolf, num=10)

def univ_step_begin(self):
    pass
```

```
def agt_init(self):
    self.x = rand() * 100
    self.y = rand() * 100
    self.direction = rand() * 360

def agt_step(self):
    # 最高速度まで加速
    if self.speed <= 3:
        self.speed += 0.1

    self.forward(self.speed)

    # 左右にランダムに揺れる
    self.turn(rand() * 60 - 30)

```

❑21.4 ループなしの空間で移動する（1）

- 「wolfsheep(A)」モデルとして保存して、実行ボタンを押しましょう
  - 10頭のwolfが縦横無尽に空間を動き回ります



空間のプロパティで「端点の処理」を「ループしない」に変えた場合どうなるでしょうか？

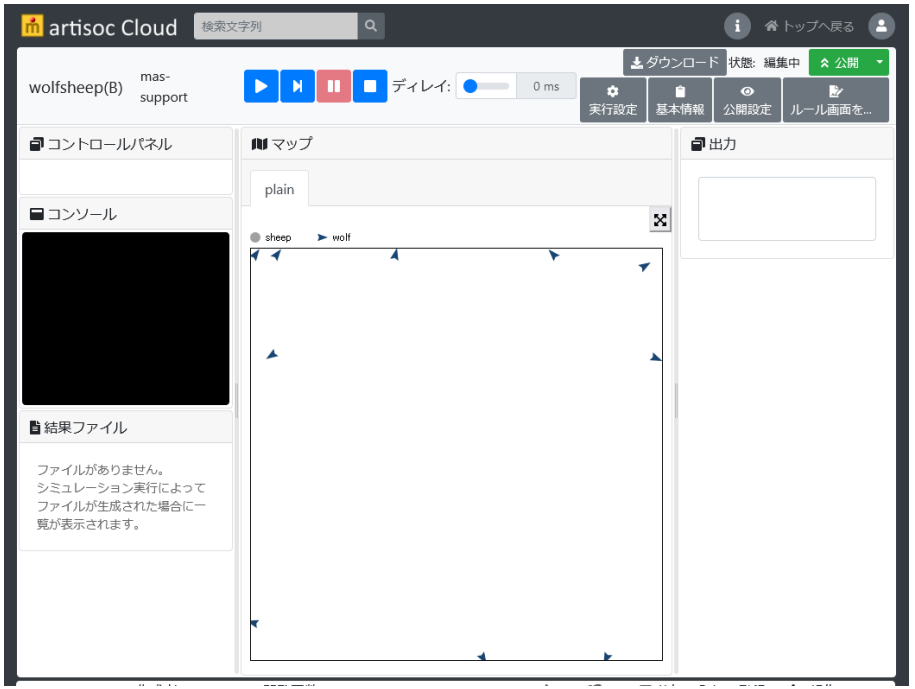
❒「wolfsheep(A)」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/XOAU56XfTjm6ewi8t29wVQ
  - モデル名「wolfsheep(A)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑21.4 ループなしの空間で移動する（2）

- 「ループしない」に変えるとwolfは壁に張り付きます
  - 「wolfsheep(B)」という名前でモデルを保存します

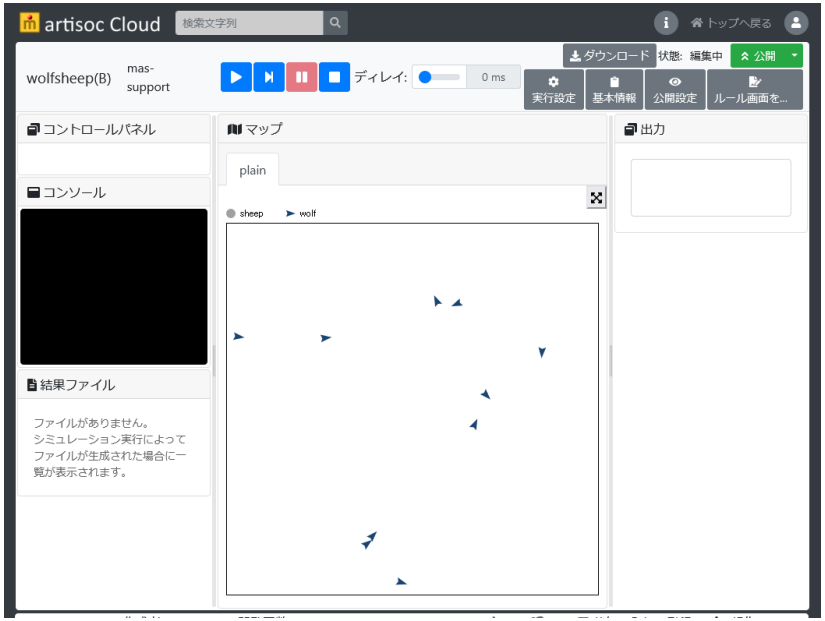


ループしない空間では，エージェントに空間の境界を認識させ，
端点に至った場合には別なルールを実行させる必要性が生じます。

❑21.4 ループなしの空間で移動する（3）

- 自分が空間の端にいるかどうかは，forward()を使って認識させることができます
  - forward()の返り値
    - 指定された距離を進めた場合：−1
    - 進めなかった場合：進めなかった分の距離を値（＝空間の端にいる）
      - ☑ 空間の端に来たら反転（つまり180度回転）する
  - wolfのルールを変更し、モデルを上書き保存した後に実行しましょう

```
1  def agt_init(self):
2      self.x = rand() * 100
3      self.y = rand() * 100
4      self.direction = rand() * 360
5
6  def agt_step(self):
7      # 最高速度まで加速
8      if self.speed <= 3:
9          self.speed += 0.1
10
11     # 前進するが，空間の端なら反対を向く
12     if self.forward(self.speed) != -1:
13         self.turn(180)
14
15     # 左右にランダムに揺れる
16     self.turn(rand() * 60 - 30)
17
```



❒「wolfsheep(B)」のモデル

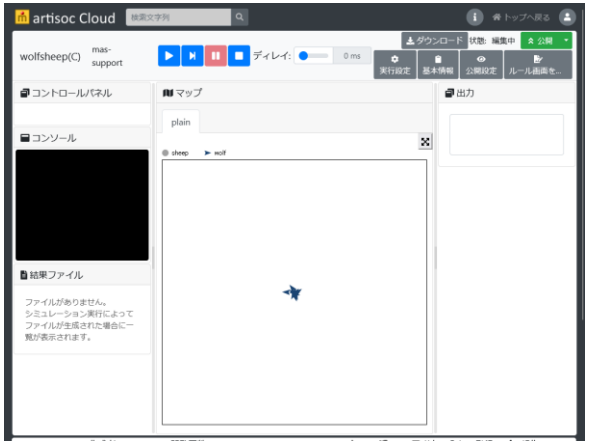
途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/spbcIwMbR6WLlH70xmT9rQ
  - モデル名「wolfsheep(B)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒21.5 目的地をめざす（1）

- wolfが草原の中央を目的地として直進する動きをルール化してみましょう
  - get_direction()：空間上のAB二地点間の角度を計算する関数
    - ➡ get_direction(AのX座標, AのY座標, BのX座標, BのY座標, 空間名)
  - 「wolfsheep(C)」という名前でモデルを保存します
    - ➡ wolfの「agt_step」にルールを追加します

```
1  def agt_init(self):
2      self.x = rand() * 100
3      self.y = rand() * 100
4      self.direction = rand() * 360
5
6  def agt_step(self):
7      # 体を目的地の方向に向ける
8      self.direction = get_direction(self.x, self.y, 50, 50,Universe.plain)
9
10     # 最高速度まで加速
11     if self.speed <= 3:
12         self.speed += 0.1
13
14     # 前進するが，空間の端なら反対を向く
15     if self.forward(self.speed) != -1:
16         self.turn(180)
17
```



wolfが草原の中央に
集まってくる

❐「wolfsheep(C)」のモデル

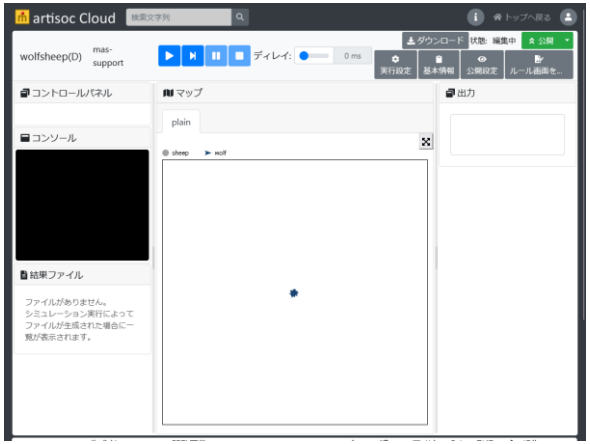
途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/G0JXade1QtmY9-q7Crp6GA
  - モデル名「wolfsheep(C)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑21.5 目的地をめざす（2）

- 草原中央に達したら静止するようルールを書き換えてみましょう
  - measure_distance()：空間上のAB二地点間の距離を求める関数
    - measure_distance(AのX座標, AのY座標, BのX座標, BのY座標, 空間名)
  - 「wolfsheep(D)」という名前でモデルを保存します
    - wolfの「agt_step」にルールを追加します

```
def agt_init(self):
    self.x = rand() * 100
    self.y = rand() * 100
    self.direction = rand() * 360

def agt_step(self):
    # 目的地の方向を向き，目的地までの距離を測る
    self.direction = get_direction(self.x, self.y, 50, 50,Universe.plain)
    distance = measure_distance(self.x, self.y, 50, 50,Universe.plain)

    # 距離に応じて速度を決める
    self.speed = 3 * (distance / 70)

    # 前進するが，空間の端なら反対を向く
    if self.forward(self.speed) != -1:
        self.turn(180)
```



中央に集まって静止する

❒「wolfsheep(D)」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/7p0ZMUMxQSeIbu3bvhAGNw
  - モデル名「wolfsheep(D)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑21.6 格子空間をランダムに移動する

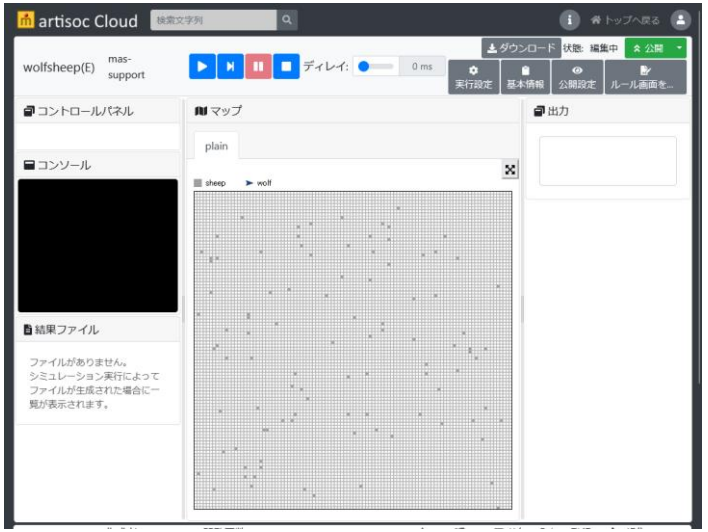
- いったんwolfの数を0、sheepの数を100にして、100匹の羊を草原に登場させましょう
  - 空間の各セルが一区画の牧草地として、離散的な空間を移動させたい
  - forward_direction_sqgrid()：格子上を指定した方向へ移動する関数
    - forward_direction_sqgrid(方向, 距離)
      - ☑ 方向　0:右、1:右上、2:上、3:左上、4:左、5:左下、6:下、7:右下

  - 「wolfsheep(E)」という名前でモデルを保存します
    - Universeのルールを更新します

```
def univ_init(self):
    # create_agt(Universe.plain.wolf, num=10)
    create_agt(Universe.plain.sheep, num=100)

def univ_step_begin(self):
    pass

def univ_step_end(self):
    pass

def univ_finish(self):
    pass
```

    - sheepのルールを更新します

```
def agt_init(self):
    self.x = int(rand() * 100)
    self.y = int(rand() * 100)

def agt_step(self):
    self.forward_direction_sqgrid(int(rand() * 8), 1)
```



sheepが一区画ずつ移動する

❒「wolfsheep(E)」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/7_S8F5Z0TzWOAfT4x_FTYg
  - モデル名「wolfsheep(E)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒21.7 追いかける（1）

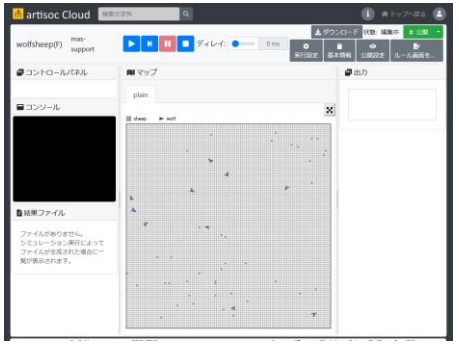
- wolfをsheepの恐ろしい天敵として再び登場させることにします
  - wolfの基本的な動き
    - 周囲10の範囲内にsheepを見つけたとき、体の向きを変え、加速をしながら直進していく
    - 標的に近接したとき、移動をやめてsheepを食べる
  - 利用する関数
    - del_agt()：エージェントを削除する関数
    - turn_agt()：目標のエージェントがいる方向に体を向ける関数
  - 「wolfsheep(F)」という名前でモデルを保存します
    - Universeのルールを更新します

```
1  def univ_init(self):
2      create_agt(Universe.plain.wolf, num=10)
3      create_agt(Universe.plain.sheep, num=100)
4
5  def univ_step_begin(self):
6      pass
7
8  def univ_step_end(self):
9      pass
10
11 def univ_finish(self):
12     pass
13
```

❑21.7 追いかける（2）

- wolfのルールを更新します

```
def agt_init(self):
    self.x = rand() * 100
    self.y = rand() * 100
    self.direction = rand() * 360

def agt_step(self):

    sheep_set = self.make_agtset_around_own(10, False, agttype=Universe.plain.sheep)

    num = count_agtset(sheep_set)

    if num == 0:        # 標的がいなければぶらぶらする
        self.turn(rand() * 60 - 30)
        self.speed = 0.5
    else:               # いれば
        one = randchoice(sheep_set)
        self.turn_agt(one)      # 体を標的の報告に向ける
        if measure_distance(self.x, self.y, one.x, one.y, Universe.plain) < 1:
            # 標的がすぐ近くなら、止まって捕食
            self.speed = 0
            del_agt(one)
        elif self.speed <= 3:   # 離れていれば、限界速度まで加速
            self.speed += 0.1

    # （捕食中以外）移動する（空間の端なら右を向く）
    if self.forward(self.speed) != -1:
        self.turn(-90)

```



sheepがどんどん
食べられていく

❒「wolfsheep(F)」のモデル

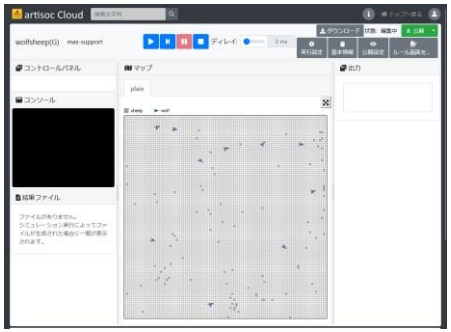
途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/JD4b6-_rQS2vtBoM9GKimw
  - モデル名「wolfsheep(F)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑21.8 逃げる

- sheepがwolfを発見した場合、正反対の方向に一目散に逃げるようにします
  - sheepの基本的な動き
    - measure_distance()を使って周囲にいるwolfとの距離を逐一チェックする
    - 最も近いwolfを選別し、get_direction()を使って角度を取得して、180度を加えることで反対方向に向きを変える
  - 「wolfsheep(G)」という名前でモデルを保存します
    - sheepのルールを更新します

```
def agt_init(self):
    self.x = int(rand() * 100)
    self.y = int(rand() * 100)

def agt_step(self):
    wolf_set = self.make_agtset_around_own(10, False, agttype=Universe.plain.wolf)
    if count_agtset(wolf_set) == 0:      # wolfがいないとき
        self.forward_direction_sqgrid(int(rand() * 8), 1)
    else:
        min_distance = 150
        for one in wolf_set:
            distance = measure_distance(self.x, self.y, one.x, one.y, Universe.plain)
            if distance < min_distance: # oneとの距離が最短なら、置き換える
                threat = one
                min_distance = distance

        deg = get_direction(self.x, self.y, threat.x, threat.y, Universe.plain) + 180
        deg_cell = int((deg + 22.5) / 45)   # 角度をセル型の方向に変換
        if deg_cell >= 8:
            deg_cell = 0

        self.forward_direction_sqgrid(deg_cell, 3)
```



sheepが逃げる

❐「wolfsheep(G)」のモデル

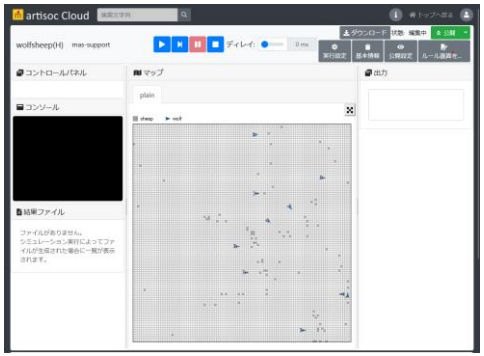
途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/7GOyYpT3Sq6wd3OzJ4ZnkA
  - モデル名「wolfsheep(G)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❒21.9 追いかけ続ける

- wolfに執拗さを加えて、一度狙いを定めたsheepをひたすら追い続けるようにします
  - wolfの基本的な動き
    - ステップを超えて同一の標的を追い続けるため、wolfに標的を保持するための変数targetを追加
  - 「wolfsheep(H)」という名前でモデルを保存します
    - wolfのルールを更新します

```
def agt_init(self):
    self.x = rand() * 100
    self.y = rand() * 100
    self.direction = rand() * 360
    self.target = set()

def agt_step(self):
    if count_agtset(self.target) == 0:  # 標的がいなければ周囲を探す
        sheep_set = self.make_agtset_around_own(10, False, agttype=Universe.plain.sheep)
        num = count_agtset(sheep_set)

        if num == 0:        # 標的がいなければぶらぶらする
            self.turn(rand() * 60 - 30)
            self.speed = 0.5
        else:               # いれば狙いをつける
            one = randchoice(sheep_set)
            self.target.add(one)    # oneを標的にリストアップ
    else:       # 既に標的が決まっていれば捕食するまで追い続ける
        one = next(iter(self.target))
        self.turn_agt(one)
        if measure_distance(self.x, self.y, one.x, one.y, Universe.plain) < 1:
            # 標的がすぐ近くなら、止まって捕食
            self.speed = 0
            self.target = set() # 標的をクリア
            del_agt(one)        # 抹消する
        elif self.speed <= 3:   # 離れていれば、限界速度まで加速
            self.speed += 0.1

    # （捕食中以外）移動する（空間の端なら右を向く）
    if self.forward(self.speed) != -1:
        self.turn(-90)
```



wolfがsheepを
追い続ける

❐「wolfsheep(H)」のモデル

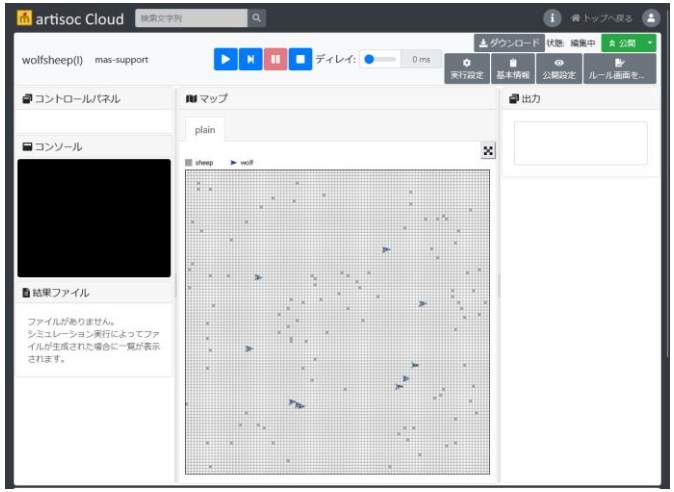
途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/DUBls　DTbavGOFqnLYmlQ
  - モデル名「wolfsheep(H)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑21.10 座標系に関連づけて移動する

- エージェントのX座標，Y座標を直接操作して移動させることもできます
  - wolfを草原の中央で円運動をさせましょう
    - wolfに変数degreeを追加
    - degreeは、エージェントの座標とX軸がなす角度
    - speedは、degreeが変化する幅、つまり1ステップあたりの角速度
  - 「wolfsheep(I)」という名前でモデルを保存します
    - wolfのルールを更新します

```
def agt_init(self):
    self.degree = 360 * rand()
    self.speed = 10 * rand()

def agt_step(self):
    self.degree += self.speed
    self.x = 50 + 30 * cos(degrees(self.degree))
    self.y = 50 + 30 * sin(degrees(self.degree))
```



wolfが円運動をする

❐「wolfsheep(I)」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/IlUujcsLT_WZxqxD95Su2A
  - モデル名「wolfsheep(I)」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑新しく学んだ事項

- ループしない空間の端点での移動の扱い
- forward()の返り値の利用
- get_direction()
- del_agt()
- turn_agt()
- measure_distance()
- 最小値の探し方（初期化、for～in文、if文、置き換え）
- 三角関数の初歩的な使い方

第22章：属性を文字列で表す

❑22.0 質的な属性を質的なままで操作できます

- 属性を表す変数は文字列が基本です
- 一つの変数でいくつもの属性を表せます
- 属性は、文字としての数字で表すと便利です
- 数字で表された属性は，色の属性に簡単に変換できます
- 属性の中身(一部分)を取り出したり，変えたりすることもできます

❑22.1 文字列で表現するとは

- 質的な属性を文字列で表すことにより，複雑な属性を扱うさまざまな技法を学ぶ
- 性別：「男」「male」，「女」「female」
- 遺伝子DNA：「AGGCTTAAC」など
- 数値の演算とは異なる文字列独自の操作方法を学ぶ
- 質的な属性を色で表す技法と文字列で表す技法とを結びつける手法も学ぶ
- 整数としての意味と文字としての意味を持ちうる数字の二重性を利用

❑22.2 文字列型の変数に慣れる(1)

- 室内パーティーのモデル
  - 広い部屋に人がたくさん歩き回っている
  - 近くに同性の人がいると、その人と同じ方向に歩き出す

- モデルの作成
  1. 新規モデル作成　モデル名: party
  2. 空間room（デフォルト）を生成
  3. その下に、エージェント種別personを作成
  4. personに変数trait（文字列で性別を表す）とview（整数で性別の色表示）
  5. univ_initでpersonを200生成
  6. マップ出力の設定：変数viewでの色指定を忘れずに

❑22.2 文字列型の変数に慣れる(2)

- personのルール（agt_init）
  - ランダム変数を利用して、男女がだいたい半々になるように
  - 文字列は“male”のように引用符（‘’でもよい）で囲む

```
def agt_init(self):
    self.x = rand() * 50
    self.y = rand() * 50
    self.direction = rand() * 360
    if rand() >= 0.5:
        self.trait = "male"
        self.view = COLOR_RED
    else:
        self.trait = "female"
        self.view = COLOR_BLUE

```

❑22.2 文字列型の変数に慣れる(3)

- personのルール（agt_step）
  - ぶらついて、近くに同性の人がいたらその人と同じ方向に歩き出す
  - 文字列が同じかどうかの比較もself.trait == one.traitのように「==」を使う

```
12 def agt_step(self):
13     self.turn(rand() * 60 - 30)
14     self.forward(1)
15     neighbor = self.make_agtset_around_own(1, False)
16     num = count_agtset(neighbor)
17     if num > 0:
18         one = randchoice(neighbor)
19         if self.trait == one.trait:
20             self.direction = one.direction
21
```

- 上書き保存して実行

❐「party」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/xCHzF1wiRPSP3JPY1H0Hnw
  - モデル名「party」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑22.3 文字列型変数も演算(操作)の対象になります(1)

- 文字列をつなげて複数の属性を1つの変数で表すことも可能
  - 最初の文字で性別(男ならm女ならf)、次の3文字で国籍(日本ならJPNアメリカならUSA)、最後の文字で働いているかどうか(有職ならe無職ならu)とすると“fJPNe”は「職に就いている日本女性」
  - 文字列は+でつなげる：self.trait = "f" + "JPN" + "e”

❑22.3 文字列型変数も演算(操作)の対象になります(2)

- 文字列から一部を取り出すことで個別の属性を比較
  - 文字列を構成する各文字には先頭から0,1,2…とインデックスが振られている
  - 各文字はtrait[index]の書式で取り出し可能：
    例）self.trait[1]→“J”, self.trait[4]→“e”
  - trait[start:end]の書式で柔軟に部分文字列を取り出し可能（endでインデックス指定した文字は含まれないので注意）：
    例）self.trait[1:4] →“JPN”
  - start, endのインデックスは省略可能：
    例）self.trait[:4] →“fJPN”, self.trait[2:] →“PNe”

❒22.4 複雑な属性を異なった色で表す(1)

- 属性を数字の文字列（ “022” “309”など ）で表し，さらにこれを色を表す整数に変換して利用

- 整数や文字列に変数の型を変換する関数int, strを利用
  - 整数から文字列へ（str()を利用） : str(7) →文字列の“7”
  - 文字列から整数へ（int()を利用） : int(7) →数としての7

❑22.4 複雑な属性を異なった色で表す(2)

- partyモデルを継承し、モデル名party Bとして保存

- エージェントは性別(男女)、食べ物の嗜好(和洋中)、飲み物の嗜好(酒、ワイン、ノンアルコール)の3属性を持つ
  - 性別を0、1、飲食の嗜好を各々0、1、2で区別
  - 属性のパターンにしたがってエージェントを異なる色で表示
  - 近くに異性のエージェントがいて、飲食の嗜好が同じだと同方向に歩き出す
  - 飲食のどちらか一方だけの嗜好が同じだと相手の嗜好に合わせる

❒22.4 複雑な属性を異なった色で表す(3)

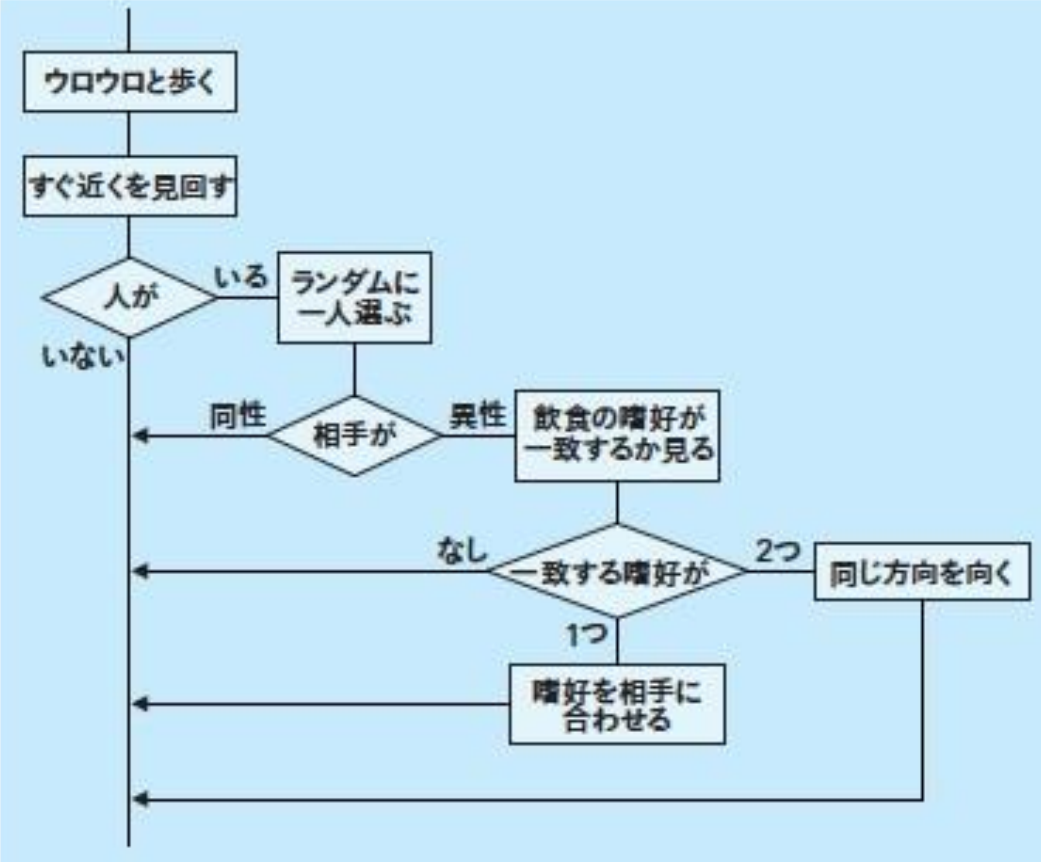
- personのルール（agt_init）

```
def agt_init(self):
    self.x = rand() * 50
    self.y = rand() * 50
    self.direction = rand() * 360

    self.trait = str(randint(0, 1)) + str(randint(0, 2)) + str(randint(0, 2))
    self.view = rgb(int(self.trait[0]) * 240, int(self.trait[1]) * 120, int(self.trait[2]) * 120)

```

- randint(m, n)はm以上n以下の整数をランダムに返す一様乱数
- str(randint(0, 1))で文字列としての“0”“1”をランダムに生成
- rgb()の引数にあるint(self.trait[0])は、生成した文字列属性traitの左端の1文字（0か1）を整数としての“0”か“1”に変換

❐22.4 複雑な属性を異なった色で表す(4)

- personのルール（agt_step）：フローチャート



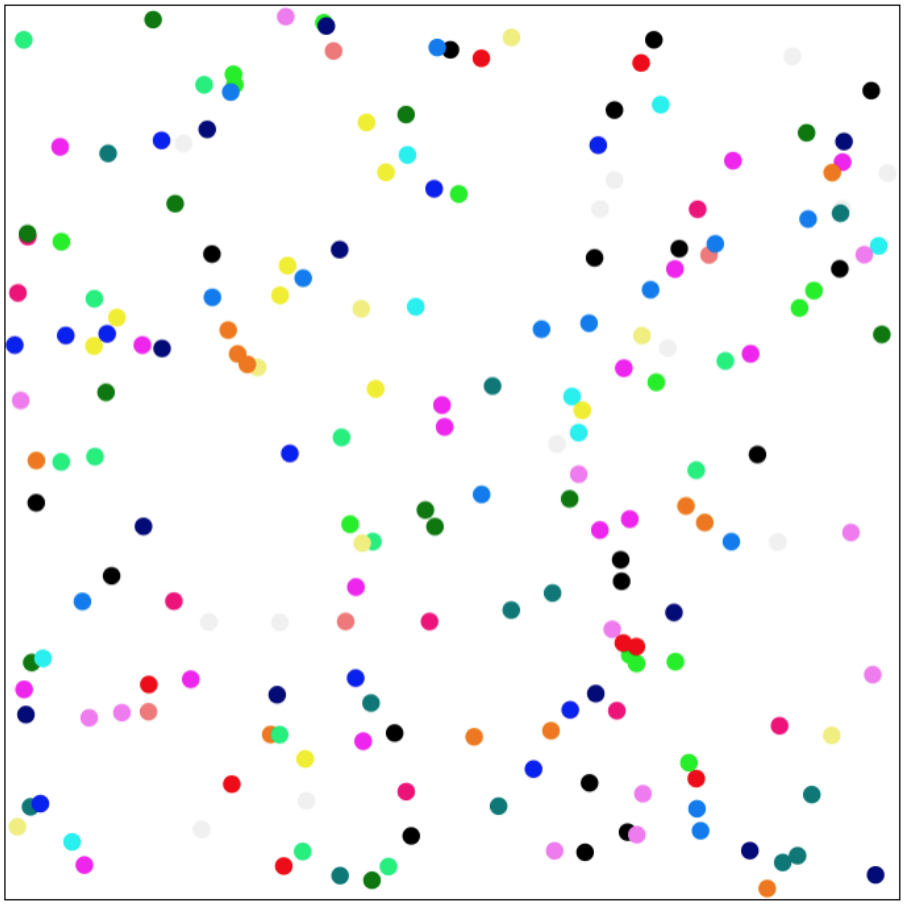
❑22.4 複雑な属性を異なった色で表す(5)

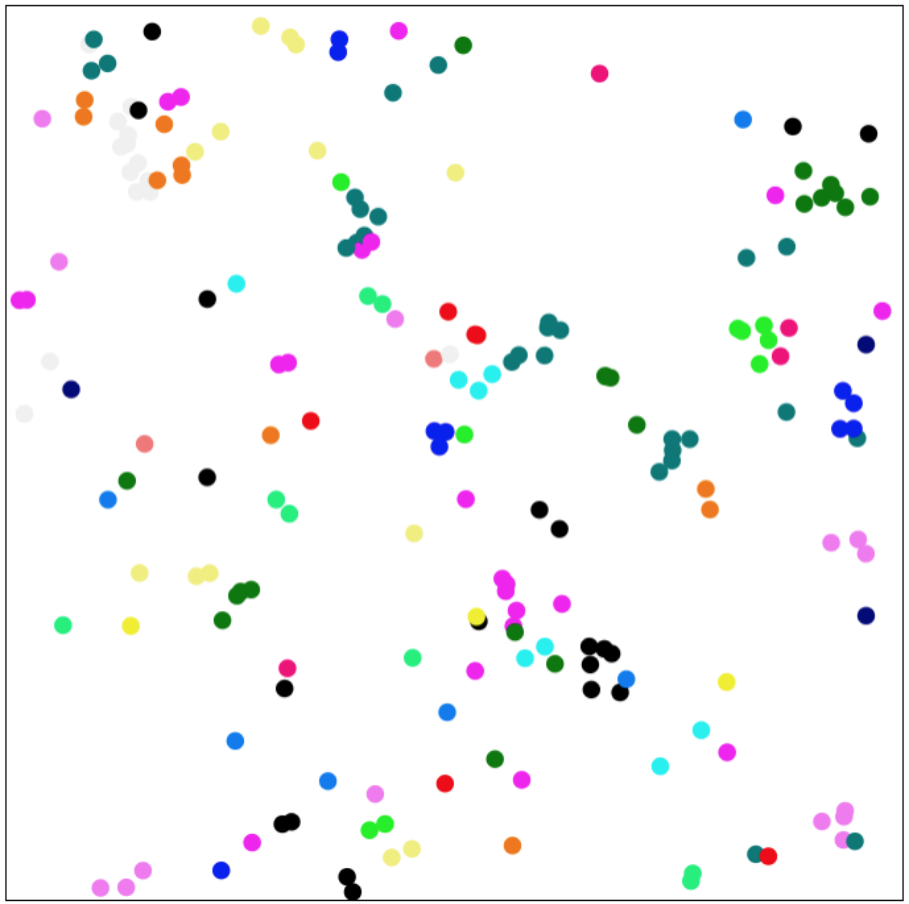
- personのルール（agt_step）

```
def agt_step(self):
    s = 0 # 嗜好の類似度sを初期化
    self.turn(rand() * 60 - 30)
    self.forward(0.5)
    neighbor = self.make_agtset_around_own(1.5, False)
    num = count_agtset(neighbor)
    if num > 0:
        one = randchoice(neighbor)
        if self.trait[0] == one.trait[0]: # 異性ならば
            for i in range(1, 3):
                if self.trait[i] == one.trait[i]:
                    s += 1
            if s == 2: # 嗜好が一致のとき
                self.direction = one.direction
            elif s == 1: # 飲食どちらかの好みが一致
                self.trait = self.trait[0] + one.trait[1:] # 相手の嗜好に合わせる
                self.view = rgb(int(self.trait[0]) * 240, int(self.trait[1]) * 120, int(self.trait[2]) * 120)

```

❐22.4 複雑な属性を異なった色で表す(6)

- エージェントの色や動く方向の変化に注目





❐「party B」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/pLkMdOlGSc2LluMyXzLtnQ
  - モデル名「party B」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑新しく学んだ事項

- 文字列型変数
- 文字列からその一部分を抽出する方法（trait[1:]など）
- 整数型変数と文字列型変数との間で型を変える関数int(), str()
- 属性と色との技巧的な関連づけ

第23章：属性を複雑に操作する:アクセルロッドの文化変容モデル（簡略版）を作る

❑23.0 周囲の状況に応じて，文化が複雑に変化します

- 前章で学んだ文字列操作法の応用が中心です。
- 文字列の操作に慣れましょう。
- 新しい技法も学びます。
- 多様性を調べる技巧的な方法を教えます。
- アクセルロッドの「文化変容」モデルの簡略版を作ります。

❑23.1 文字列の書き換えを文化変容に対応させる(1)

- 人工社会で属性を用いるモデルの典型は文化をめぐる相互作用
  - 特定の文化(属性の組み合わせ)をもった集団(共同体)が周囲の集団と相互作用
  - 相互作用する中で文化の伝播，流布，均質化などが進む

- 文化をめぐる相互作用をモデル化したものとして有名なロバート・アクセルロッドの「文化変容」モデルを作る

❑23.1 文字列の書き換えを文化変容に対応させる(2)

- アクセルロッドのモデルの簡略版
  - ある社会には，文化的に多様な共同体が共存している
  - 周囲の共同体との文化の類似性に応じて，自分の文化が周囲の共同体の文化に似てくる

1. 新規モデル作成　モデル名: dissemination A
2. 空間country（空間種別は四角格子空間；大きさは10×10）を生成
3. その下に、エージェント種別communityを作成
4. communityに変数trait（文字列で文化を表す）とview（整数で文化に応じた色を表示）
5. マップ出力の設定：変数viewの色指定を忘れずに

❒23.2 エージェントの初期配置(1)

- エージェントの生成と配置（univ_init)
  - 格子空間にエージェントを敷き詰める
  - 空間の大きさ(幅と高さ)をルールの中で測定する関数get_width_space(), get_height_space()を用いて，空間の大きさ変更に対応できるようにする

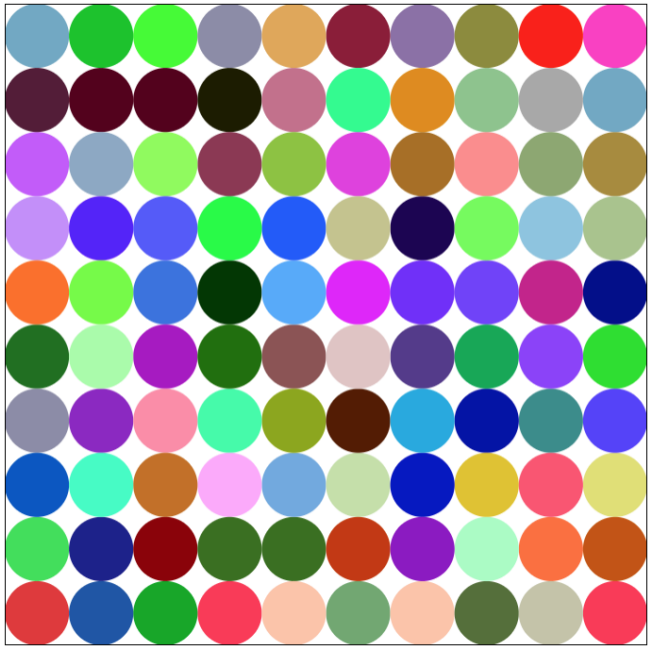
```
def univ_init(self):
    num = get_width_space(Universe.country) * get_height_space(Universe.country)
    create_agt(Universe.country.community, num=num)
    one = make_agtset(space=Universe.country)
    random_put_agtset_sqgrid(one, False)
```

❑23.2 エージェントの初期配置(2)

- エージェントに属性と色を割り当て（agt_init）
  - 文化は3つの側面(文化要素)から構成され，各々0から9までの10種類の異なるタイプからなると仮定
  - 属性はランダムに割り振る

```
1 def agt_init(self):
2     self.trait = str(randint(0, 9)) + str(randint(0, 9)) + str(randint(0, 9))
3     self.view = rgb(int(self.trait[0]) * 28, int(self.trait[1]) * 28, int(self.trait[2]) * 28)
```

- 上書き保存をして実行ボタンを押す



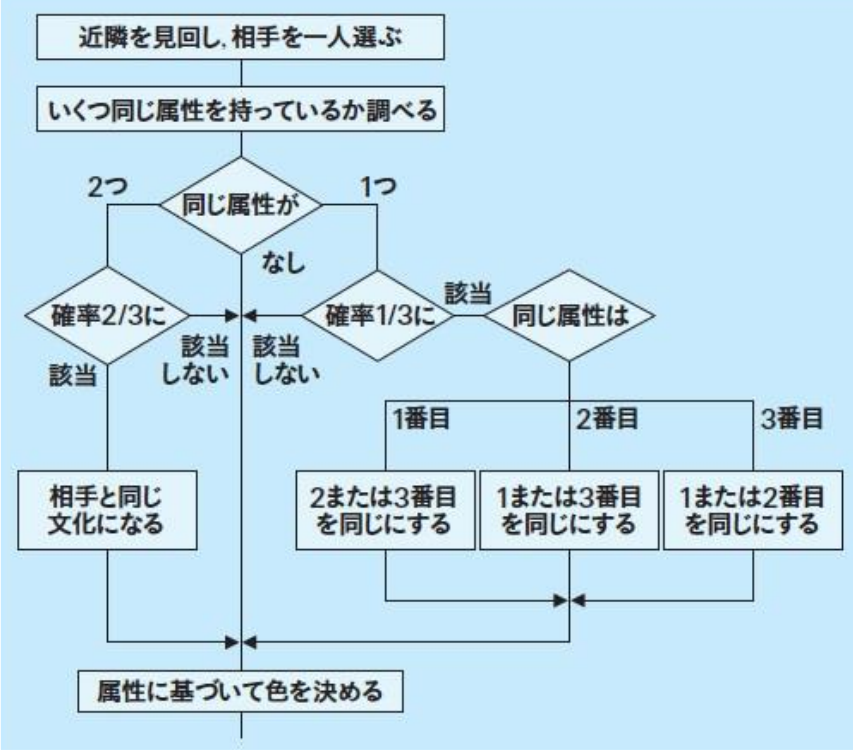
❒「dissemination A」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/lM-3wWRoQTudi-F2xbyKPA
  - モデル名「dissemination A」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑23.3 周囲のエージェントの属性に応じて自分の属性を変える(1)

- エージェントの属性変化のルールのポイント
  - 近くにいるエージェントとの文化の類似度を判断すること
  - 類似度の違いに応じて文化変容のパターンが異なる
    （文化が似ているほど，真似る確率も高くなる）



❑23.3 周囲のエージェントの属性に応じて自分の属性を変える(2)

- communityのルール（agt_step）

```
5  def agt_step(self):
6      s = 0
7      neighbor = self.make_agtset_around_own_sqgrid(1, False)
8      one = randchoice(neighbor) # 近隣から一つ選ぶ
9      for i in range(3):
10         if self.trait[i] == one.trait[i]:
11             s += 1
12     if s == 2 and rand() * 3 >= 1: # 類似度2なら確率2/3で変化
13         self.trait = one.trait
14         self.view = one.view
```

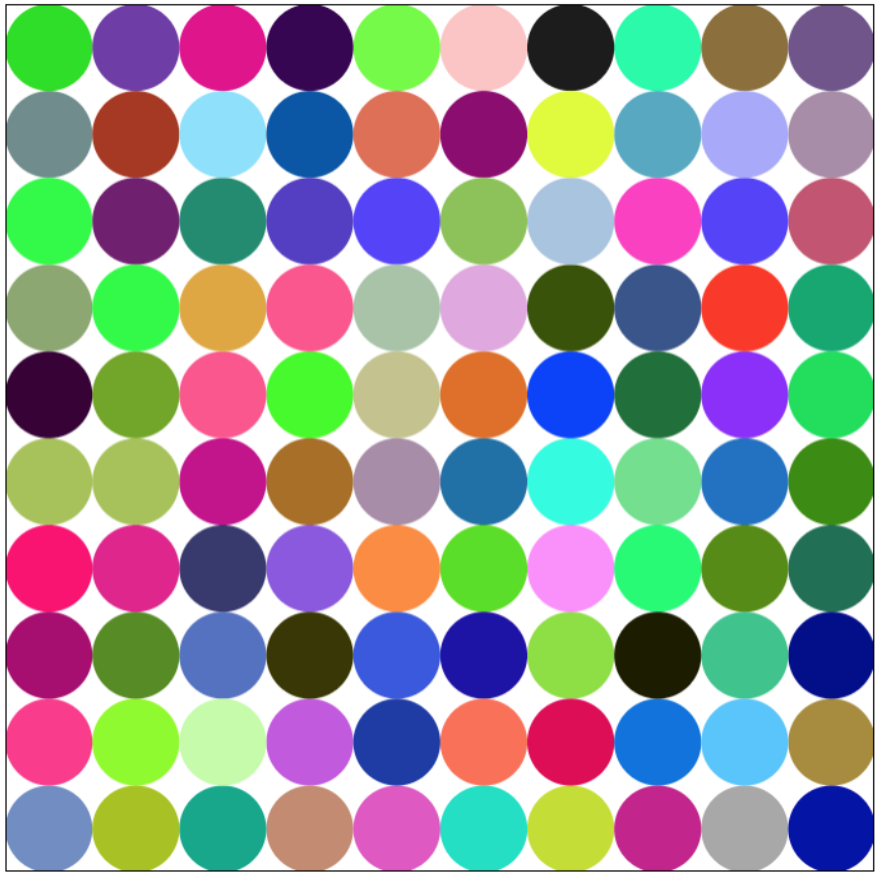
❑23.3 周囲のエージェントの属性に応じて自分の属性を変える(3)

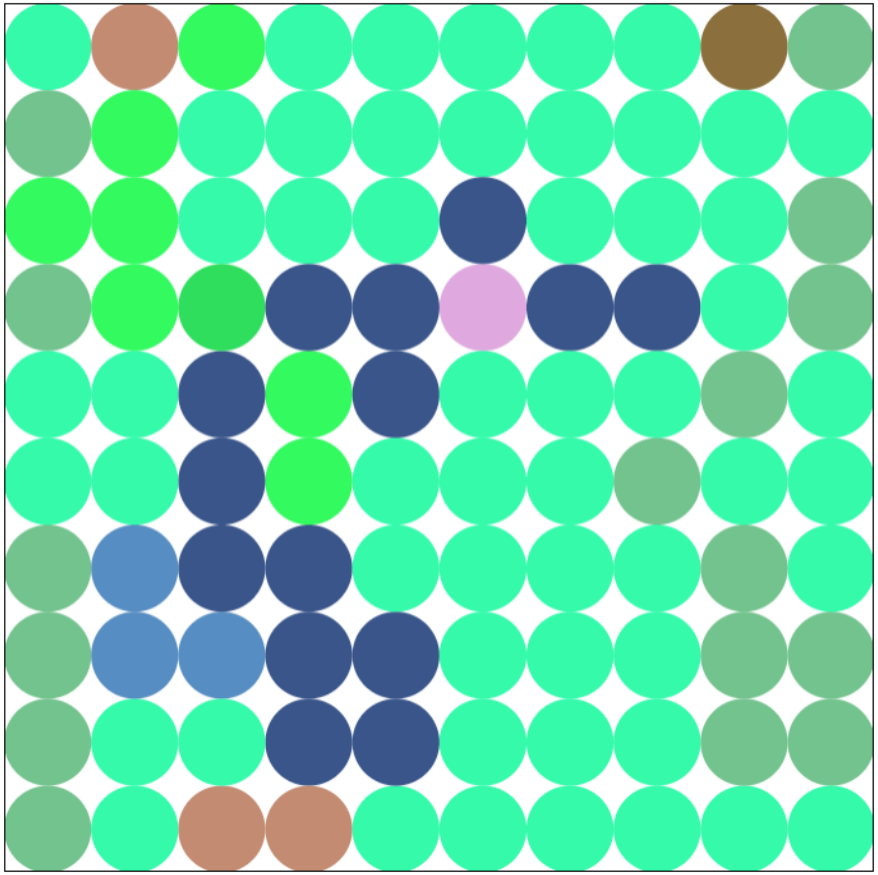
- communityのルール（続き）

```
    elif s == 1 and rand() * 3 >= 2: # 類似度1なら確率1/3で変化
        if self.trait[0] == one.trait[0]: # 最初の属性が同じ
            if rand() <= 0.5:
                self.trait = self.trait[0] + one.trait[1] + self.trait[2]
            else:
                self.trait = self.trait[0] + self.trait[1] + one.trait[2]
        elif self.trait[1] == one.trait[1]: # 2番目の属性が同じ
            if rand() <= 0.5:
                self.trait = one.trait[0] + self.trait[1] + self.trait[2]
            else:
                self.trait = self.trait[0] + self.trait[1] + one.trait[2]
        else: # 最後の属性が同じ
            if rand() <= 0.5:
                self.trait = one.trait[0] + self.trait[1] + self.trait[2]
            else:
                self.trait = self.trait[0] + one.trait[1] + self.trait[2]
    self.view = rgb(int(self.trait[0]) * 28, int(self.trait[1]) * 28, int(self.trait[2]) * 28)
```

❑23.3 周囲のエージェントの属性に応じて自分の属性を変える(4)

- モデル名を「dissemination B」に変更して保存し、実行





❒「dissemination B」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/YE_GX-8lQ4OHOuyDwmLqgA
  - モデル名「dissemination B」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑23.4 オリジナルとの違い

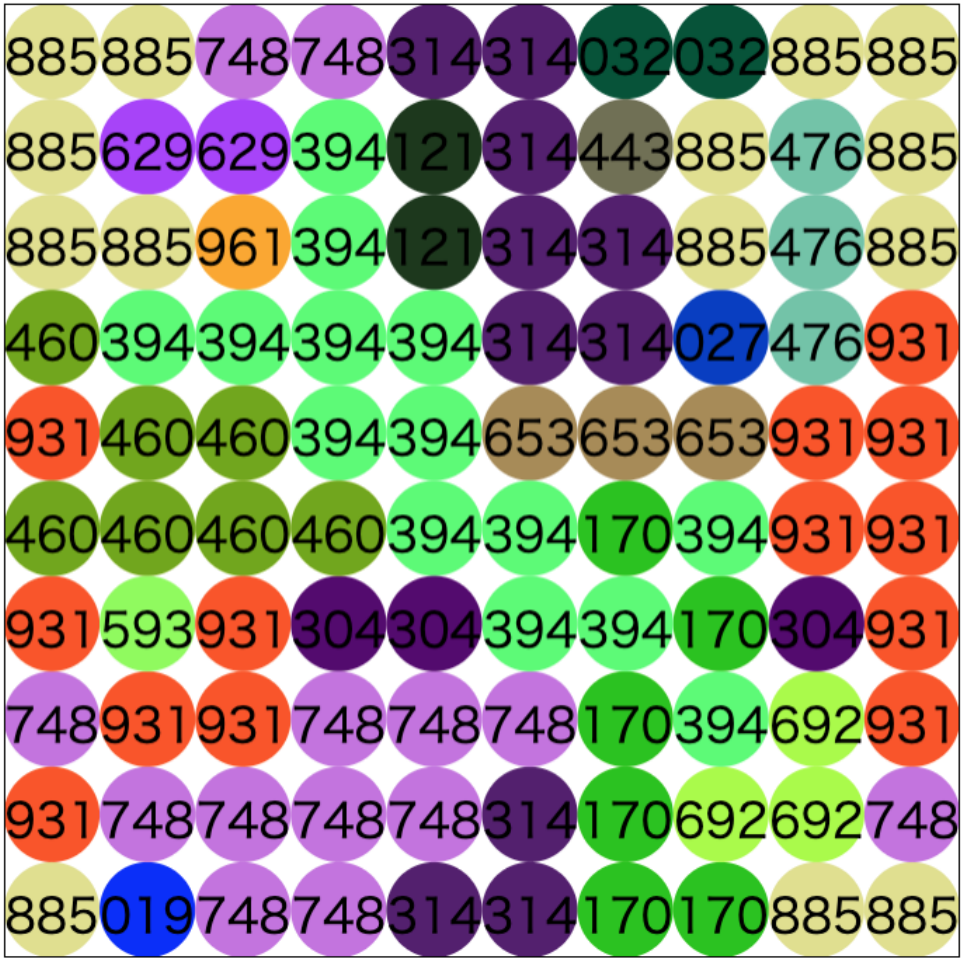
- アクセルロッドの文化変容モデルの発想法を忠実に再現
- 他方で3点で簡略化
  - アクセルロッドのモデルでは文化を表す属性が3つではなく，5つ
  - アクセルロッドのモデルでは，近隣にいるエージェントは，8方向にいるエージェント（「ムーア近傍」）ではなく，東西南北の4方向にいるエージェントだけ（「ノイマン近傍」）
  - アクセルロッドのモデルでは，空間はループしていない
- どれも本質的な違いはない

❑23.5 属性を直接マップに示す(1)

- エージェントの変数の値を直接マップの上に表示させることが可能
- マップ出力
  - マップ出力設定＞マップ要素設定（エージェント）
  - 「エージェント情報の表示」の「表示する変数」でtraitを選択

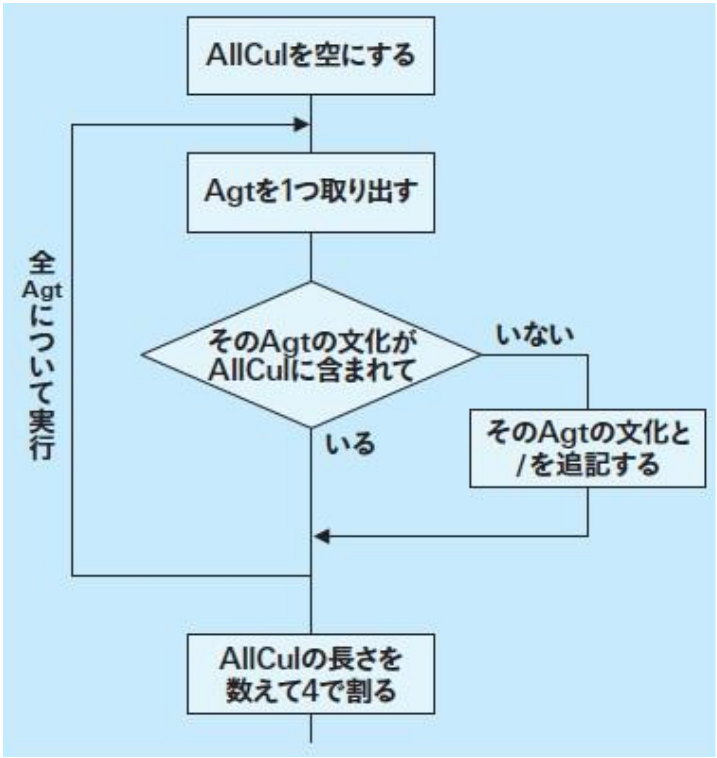
❑23.5 属性を直接マップに示す(2)

- モデル名を「dissemination C」に変更して保存し、実行



❑23.6 全体の様子を把握する(1)

- エージェントがさまざまなタイプの文化(属性の組み合わせ)を持っているときに，全部で何パターン存在しているのか調べる。
  - Universeに文化の数を記録する整数型変数numculを追加
  - numculを時系列グラフに出力するよう設定
  - univ_step_endで必要な集計作業（フローチャート参照）



❑23.6 全体の様子を把握する(2)

- 文化パターンの数の算出（univ_step_end）

```
10  def univ_step_end(self):
11      allcul = "" # 文字列型変数の初期化
12      total = make_agtset(agttype=Universe.country.community)
13      for one in total:
14          if one.trait not in allcul:
15              allcul = allcul + one.trait + "/"
16      Universe.numcul = len(allcul) / 4
17
18  def univ_finish(self):
19      pass
```

- 文化のパターンを記録する文字列型変数allcul
- 個々のエージェントの文化（one.trait）がallculに含まれていないかを演算子not inを用いて判断：
  one.traitがallculの部分文字列でないならone.trait not in allcul→True
- allcul = allcul + one.trait + “/”のスラッシュ（/）は文化パターンの混同を防ぐ切れ目として必要
- len()は文字列型変数を引数に取る場合その長さ（文字数）を返す

❐「dissemination C」のモデル

途中でついていけなくなった人・モデルがうまく動かない人

- 下記のモデルにアクセスしモデルを継承してください
  - https://artisoc-cloud.kke.co.jp/models/CwclQ8CkTKWfOBIxN126fw
  - モデル名「dissemination C」で公開しています
  - 継承：他のユーザが作ったモデルをコピーして、自分のモデルを新しく作る

❑新しく学んだ事項

- 空間の大きさを変数として表す関数get_width_space(), get_height_space()
- ある文字列が文字列型変数の中に存在しているか調べる演算子not in
- 文字列の長さを調べる関数len()
- エージェントの変数の値をマップ上に表示する